Apache Spark & Scala

By: Kaushik Prasad Dey

**Bicycle Sharing Demand**

**Domain –Transportation Industry**

**Business challenge/requirement**

With the spike in pollution levels and the fuel prices, many Bicycle Sharing Programs are running around the world. Bicycle sharing systems are a means of renting bicycles where the process of obtaining membership, rental and bike return is automated via a network of joint locations throughout the city. Using this system people can rent a bike from one location and return it to a different place as and when needed.

**Data Set**

Data contains hourly rental data spanning two years. Training set comprised of the first 19 days of each month while the test set is the 20th to the end of month.

**Considerations**

You are building a Bicycle Sharing demand forecasting service that combines historical usage patterns with weather data to forecast the Bicycle rental demand in real-time. To develop this system, you must first explore the dataset and build a model. Once it’s done you must persist the model and then on each request run a Spark job to load the model and make predictions on each Spark Streaming request.

**Data Exploration and Transformation**

**Explore the data and develop the model in SparkShell**

**import all spark dependencies**

**===========================================================================**

**import org.apache.spark.rdd.RDD**

**import org.apache.spark.ml.Pipeline**

**import org.apache.spark.ml.feature.StringIndexer**

**import org.apache.spark.ml.feature.VectorAssembler**

**import org.apache.spark.util.IntParam**

**import org.apache.spark.sql.SQLContext**

**import org.apache.spark.sql.functions.\_**

**import org.apache.spark.sql.\_**

**import org.apache.spark.sql.SparkSession**

**import org.apache.spark.sql.types.\_**

**import org.apache.log4j.\_**

**import org.apache.spark.sql.functions.to\_timestamp**

**import org.apache.spark.ml.regression.LinearRegression**

**import org.apache.spark.ml.evaluation.RegressionEvaluator**

**import org.apache.spark.ml.feature.OneHotEncoder**

**import org.apache.spark.ml.regression.{GBTRegressionModel, GBTRegressor}**

**import org.apache.spark.ml.regression.DecisionTreeRegressor**

**import org.apache.spark.ml.regression.RandomForestRegressor**

**import org.apache.spark.ml.\_**

**import org.apache.spark.rdd.RDD**

**import org.apache.spark.ml.Pipeline**

**import org.apache.spark.ml.feature.StringIndexer**

**import org.apache.spark.ml.feature.VectorAssembler**

**import org.apache.spark.util.IntParam**

**import org.apache.spark.sql.SQLContext**

**import org.apache.spark.sql.functions.\_**

**import org.apache.spark.sql.\_**

**import org.apache.spark.sql.SparkSession**

**import org.apache.spark.sql.types.\_**

**import org.apache.log4j.\_**

**import org.apache.spark.sql.functions.to\_timestamp**

**import org.apache.spark.ml.regression.LinearRegression**

**import org.apache.spark.ml.evaluation.RegressionEvaluator**

**import org.apache.spark.ml.feature.OneHotEncoder**

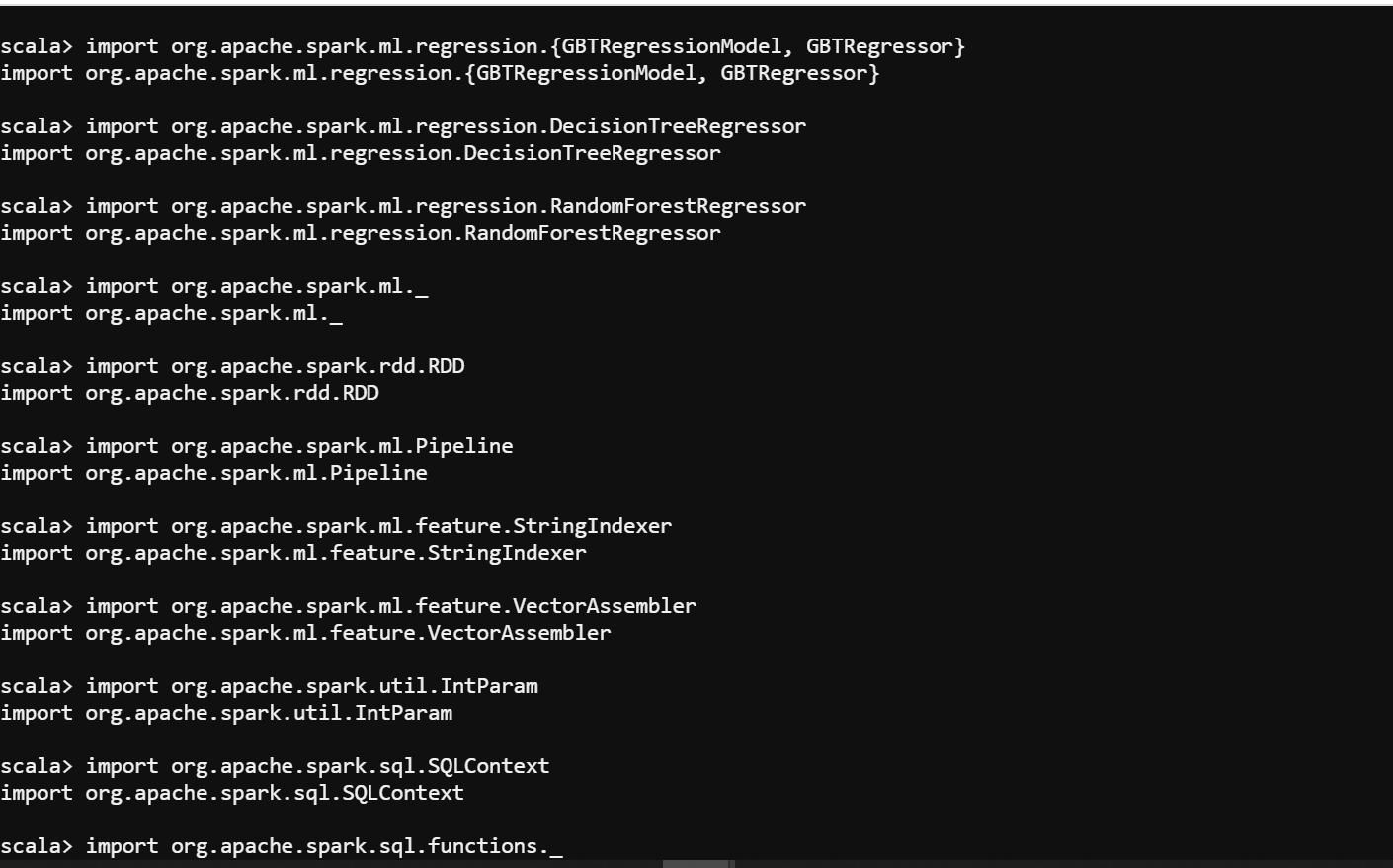
**import org.apache.spark.ml.regression.{GBTRegressionModel, GBTRegressor}**

**import org.apache.spark.ml.regression.DecisionTreeRegressor**

**import org.apache.spark.ml.regression.RandomForestRegressor**

**import org.apache.spark.ml.\_**

**===========================================================================**



* 1. **Load Dataset in Spark:**

val spark=SparkSession.builder().appName("Bicycle\_Sharing\_System\_Analysis").getOrCreate()

val train\_dataset\_df = spark.read.format("csv").option("inferSchema",true).option("header",true).load("/user/kaushikdey1984yahoo/BikeSharingApplication/train.csv")

**1.2 Read Dataset in Spark:**

train\_dataset\_df. show(10)

A picture containing text

Description automatically generated

==================================================================================

**2.Get summary of data and variable types**

**2.1 Print Schema in Spark:**

train\_dataset\_df.printSchema

root |-- datetime: string (nullable = true) |-- season: integer (nullable = true) |-- holiday: integer (nullable = true) |-- workingday: integer (nullable = true) |-- weather: integer (nullable = true) |-- temp: double (nullable = true) |-- atemp: double (nullable = true) |-- humidity: integer (nullable = true) |-- windspeed: double (nullable = true) |-- casual: integer (nullable = true) |-- registered: integer (nullable = true) |-- count: integer (nullable = true)

===============================================================================

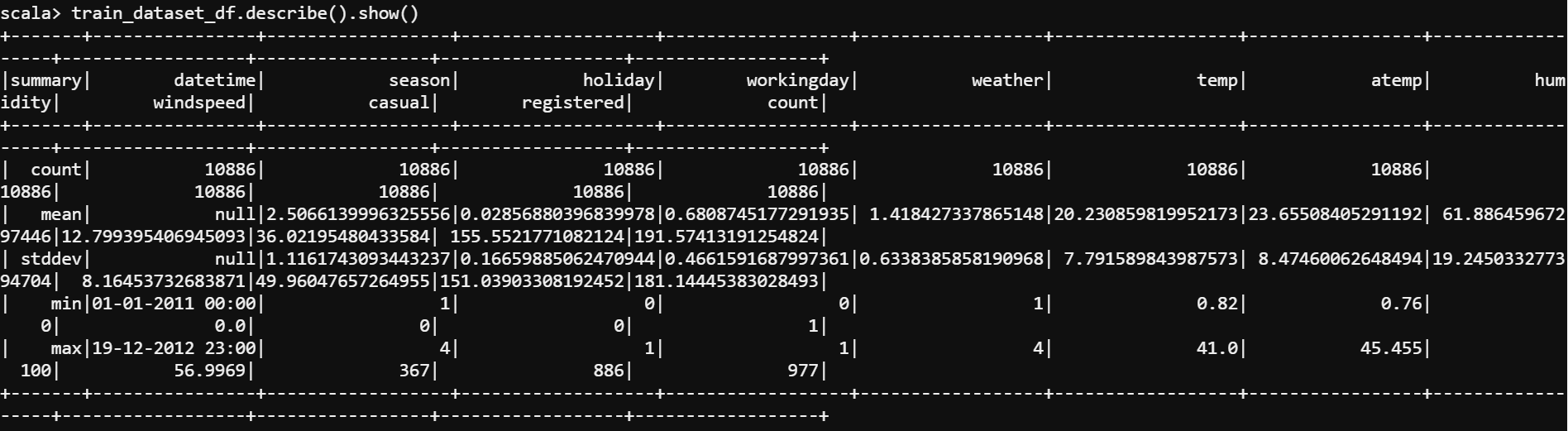
Text

Description automatically generated

* 1. **Five Point Summary of that data:**

train\_dataset\_df.describe().show()

+-------+----------------+------------------+-------------------+------------------+------------------+------------------+-----------------+------------------+------------------+-----------------+------------------+------------------+|summary| datetime| season| holiday| workingday| weather| temp| atemp| humidity| windspeed| casual| registered| count|+-------+----------------+------------------+-------------------+------------------+------------------+------------------+-----------------+------------------+------------------+-----------------+------------------+------------------+| count| 10886| 10886| 10886| 10886| 10886| 10886| 10886| 10886| 10886| 10886| 10886| 10886|| mean| null|2.5066139996325556|0.02856880396839978|0.6808745177291935| 1.418427337865148|20.230859819952173|23.65508405291192| 61.88645967297446|12.799395406945093|36.02195480433584| 155.5521771082124|191.57413191254824|| stddev| null|1.1161743093443237|0.16659885062470944|0.4661591687997361|0.6338385858190968| 7.791589843987573| 8.47460062648494|19.245033277394704| 8.16453732683871|49.96047657264955|151.03903308192452|181.14445383028493|| min|01-01-2011 00:00| 1| 0| 0| 1| 0.82| 0.76| 0| 0.0| 0| 0| 1|| max|19-12-2012 23:00| 4| 1| 1| 4| 41.0| 45.455| 100| 56.9969| 367| 886| 977|+-------+----------------+------------------+-------------------+------------------+------------------+------------------+-----------------+------------------+------------------+-----------------+------------------+------------------+



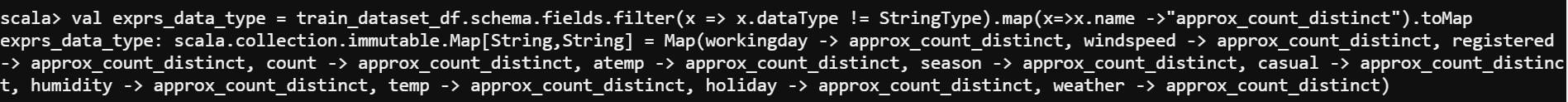
===============================================================================

**3.Decide which columns should be categorical and then convert them accordingly**

val exprs\_data\_type = train\_dataset\_df.schema.fields.filter(x => x.dataType != StringType).map(x=>x.name ->"approx\_count\_distinct").toMap

**output:**

exprs\_data\_type: scala.collection.immutable.Map[String,String] = Map(workingday -> approx\_count\_distinct, windspeed -> approx\_count\_distinct, registered -> approx\_count\_distinct, count -> approx\_count\_distinct, atemp -> approx\_count\_distinct, season -> approx\_count\_distinct, casual -> approx\_count\_distinct, humidity -> approx\_count\_distinct, temp -> approx\_count\_distinct, holiday -> approx\_count\_distinct, weather -> approx\_count\_distinct)



**Display the aggregated data**

train\_dataset\_df.agg(exprs\_data\_type).show()

23/01/09 19:21:06 WARN util.Utils: Truncated the string representation of a plan since it was too large. This behavior can be adjusted by setting 'spark.debug.maxToStringFields' in SparkEnv.conf.+---------------------------------+--------------------------------+---------------------------------+----------------------------+----------------------------+-----------------------------+-----------------------------+-------------------------------+---------------------------+------------------------------+------------------------------+|approx\_count\_distinct(workingday)|approx\_count\_distinct(windspeed)|approx\_count\_distinct(registered)|approx\_count\_distinct(count)|approx\_count\_distinct(atemp)|approx\_count\_distinct(season)|approx\_count\_distinct(casual)|approx\_count\_distinct(humidity)|approx\_count\_distinct(temp)|approx\_count\_distinct(holiday)|approx\_count\_distinct(weather)|+---------------------------------+--------------------------------+---------------------------------+----------------------------+----------------------------+-----------------------------+-----------------------------+-------------------------------+---------------------------+------------------------------+------------------------------+| 2| 27| 726| 802| 60| 4| 311| 88| 49| 2| 4|+---------------------------------+--------------------------------+---------------------------------+----------------------------+----------------------------+-----------------------------+-----------------------------+-------------------------------+---------------------------+------------------------------+------------------------------+

Graphical user interface

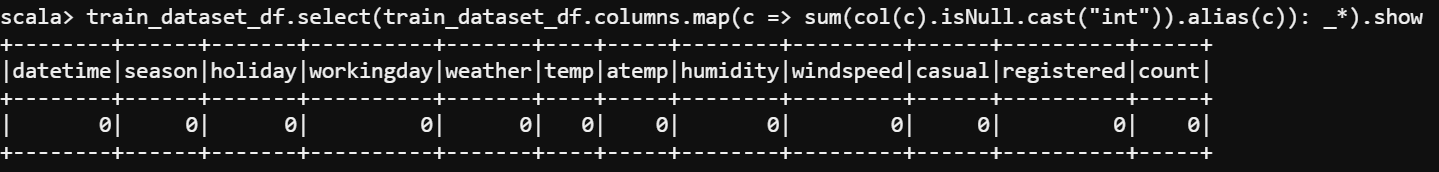
Description automatically generated

==================================================================================

**4.** **Check for any missing value in dataset and treat it**

train\_dataset\_df.select(train\_dataset\_df.columns.map(c => sum(col(c).isNull.cast("int")).alias(c)): \_\*).show

+--------+------+-------+----------+-------+----+-----+--------+---------+------+----------+-----+|datetime|season|holiday|workingday|weather|temp|atemp|humidity|windspeed|casual|registered|count|+--------+------+-------+----------+-------+----+-----+--------+---------+------+----------+-----+| 0| 0| 0| 0| 0| 0| 0| 0| 0| 0| 0| 0|+--------+------+-------+----------+-------+----+-----+--------+---------+------+----------+-----+



**5. Explode season column into separate columns such as season\_<val> and drop season**

scala>val indexer = Array("season").map(c=>new OneHotEncoder().setInputCol(c).setOutputCol(c + "\_val"))

scala> val pipeline = new Pipeline().setStages(indexer)

pipeline: org.apache.spark.ml.Pipeline = pipeline\_ffad05951e06

scala> val season\_data\_column = pipeline.fit(train\_dataset\_df).transform(train\_dataset\_df).drop("season")

season\_data\_column: org.apache.spark.sql.DataFrame = [datetime: string, holiday: int ... 10 more fields]

scala> season\_data\_column.show(2)

+----------------+-------+----------+-------+----+------+--------+---------+------+----------+-----+-------------+

| datetime|holiday|workingday|weather|temp| atemp|humidity|windspeed|casual|registered|count| season\_val|

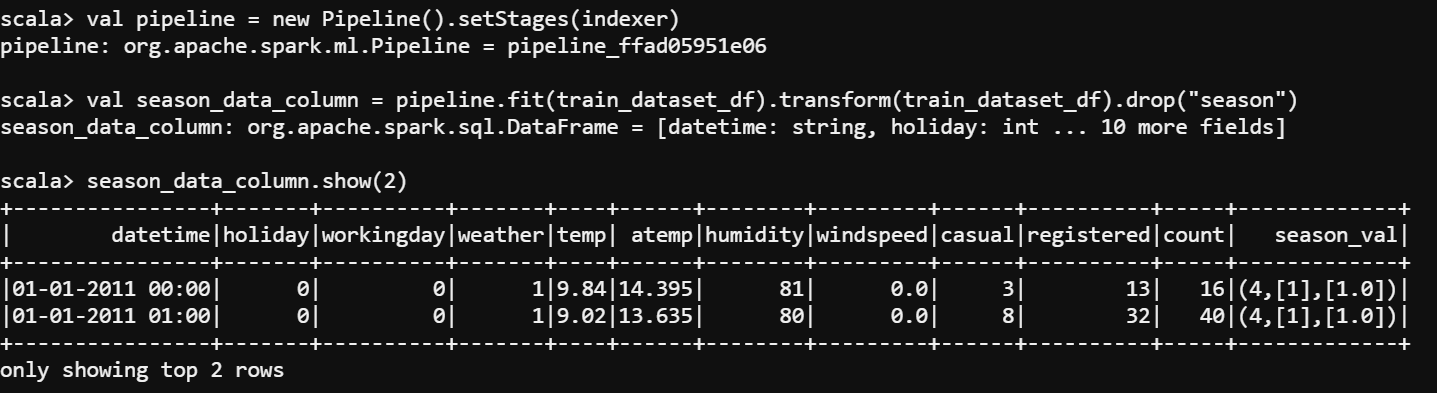
+----------------+-------+----------+-------+----+------+--------+---------+------+----------+-----+-------------+

|01-01-2011 00:00| 0| 0| 1|9.84|14.395| 81| 0.0| 3| 13| 16|(4,[1],[1.0])|

|01-01-2011 01:00| 0| 0| 1|9.02|13.635| 80| 0.0| 8| 32| 40|(4,[1],[1.0])|

+----------------+-------+----------+-------+----+------+--------+---------+------+----------+-----+-------------+

only showing top 2 rows



==================================================================================

**6. Execute the same for weather as weather\_<val> and drop weather**

scala> val indexer = Array("weather").map(c=>new OneHotEncoder().setInputCol(c).setOutputCol(c + "\_val"))

warning: there was one deprecation warning; re-run with -deprecation for details

indexer: Array[org.apache.spark.ml.feature.OneHotEncoder] = Array(oneHot\_15b8defed343)

scala> val pipeline = new Pipeline().setStages(indexer)

pipeline: org.apache.spark.ml.Pipeline = pipeline\_859d8df82084

scala> val weather\_data\_column = pipeline.fit(train\_dataset\_df).transform(train\_dataset\_df).drop("weather")

weather\_data\_column: org.apache.spark.sql.DataFrame = [datetime: string, season: int ... 10 more fields]

scala> weather\_data\_column.show(2)

+----------------+------+-------+----------+----+------+--------+---------+------+----------+-----+-------------+

| datetime|season|holiday|workingday|temp| atemp|humidity|windspeed|casual|registered|count| weather\_val|

+----------------+------+-------+----------+----+------+--------+---------+------+----------+-----+-------------+

|01-01-2011 00:00| 1| 0| 0|9.84|14.395| 81| 0.0| 3| 13| 16|(4,[1],[1.0])|

|01-01-2011 01:00| 1| 0| 0|9.02|13.635| 80| 0.0| 8| 32| 40|(4,[1],[1.0])|

+----------------+------+-------+----------+----+------+--------+---------+------+----------+-----+-------------+

only showing top 2 rows

=================================================================================

**6a. Drop the both data column and merge them together**

scala> val indexer = Array("season","weather").map(c=>new OneHotEncoder().setInputCol(c).setOutputCol(c + "\_val"))

warning: there was one deprecation warning; re-run with -deprecation for details

indexer: Array[org.apache.spark.ml.feature.OneHotEncoder] = Array(oneHot\_8c244e68bfcc, oneHot\_616366e2fd06)

scala> val pipeline = new Pipeline().setStages(indexer)

pipeline: org.apache.spark.ml.Pipeline = pipeline\_d9fa7a7be1ca

scala> val season\_weather\_data\_column = pipeline.fit(train\_dataset\_df).transform(train\_dataset\_df).drop("season","weather")

season\_weather\_data\_column: org.apache.spark.sql.DataFrame = [datetime: string, holiday: int ... 10 more fields]

scala> season\_weather\_data\_column.show(2)

+----------------+-------+----------+----+------+--------+---------+------+----------+-----+-------------+-------------+

| datetime|holiday|workingday|temp| atemp|humidity|windspeed|casual|registered|count| season\_val| weather\_val|

+----------------+-------+----------+----+------+--------+---------+------+----------+-----+-------------+-------------+

|01-01-2011 00:00| 0| 0|9.84|14.395| 81| 0.0| 3| 13| 16|(4,[1],[1.0])|(4,[1],[1.0])|

|01-01-2011 01:00| 0| 0|9.02|13.635| 80| 0.0| 8| 32| 40|(4,[1],[1.0])|(4,[1],[1.0])|

+----------------+-------+----------+----+------+--------+---------+------+----------+-----+-------------+-------------+

only showing top 2 rows

==================================================================================

**7. Split datetime into meaning columns such as hour, day, month, year, etc.**

scala> val dataframe\_date\_time\_season\_wather = season\_weather\_data\_column.withColumn("datetime", to\_timestamp(col("datetime"),"d-M-y H:m"))

dataframe\_date\_time\_season\_wather: org.apache.spark.sql.DataFrame = [datetime: timestamp, holiday: int ... 10 more fields]

scala> val dataframe\_date\_time\_season\_wather\_trainDF = dataframe\_date\_time\_season\_wather.withColumn("year", year(col("datetime"))).withColumn("month", mo

nth(col("datetime"))).withColumn("day", dayofmonth(col("datetime"))).withColumn("hour", hour(col("datetime"))).withColumn("minute",minute(col("datetime")

))

dataframe\_date\_time\_season\_wather\_trainDF: org.apache.spark.sql.DataFrame = [datetime: timestamp, holiday: int ... 15 more fields]

scala> dataframe\_date\_time\_season\_wather\_trainDF.select("year","month","day","hour","minute").show(5)

+----+-----+---+----+------+

|year|month|day|hour|minute|

+----+-----+---+----+------+

|2011| 1| 1| 0| 0|

|2011| 1| 1| 1| 0|

|2011| 1| 1| 2| 0|

|2011| 1| 1| 3| 0|

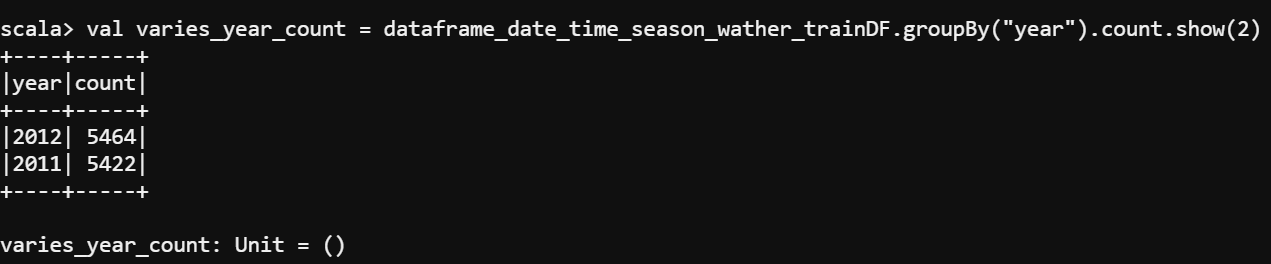
|2011| 1| 1| 4| 0|

+----+-----+---+----+------+

only showing top 5 rows

**8. Explore how count varies with different features such as hour, month, etc.**

scala> val varies\_year\_count = dataframe\_date\_time\_season\_wather\_trainDF.groupBy("year").count.show(2)+----+-----+ |year|count|+----+-----+|2012| 5464||2011| 5422|+----+-----+varies\_year\_count: Unit = ()



scala> val varies\_month\_count = dataframe\_date\_time\_season\_wather\_trainDF.groupBy("month").count.show(2)+-----+-----+ |month|count|+-----+-----+| 12| 912|| 1| 884|+-----+-----+only showing top 2 rowsvaries\_month\_count: Unit = ()

Text

Description automatically generated

==================================================================================

**Model Development**

1.Split the dataset into train and train\_test model

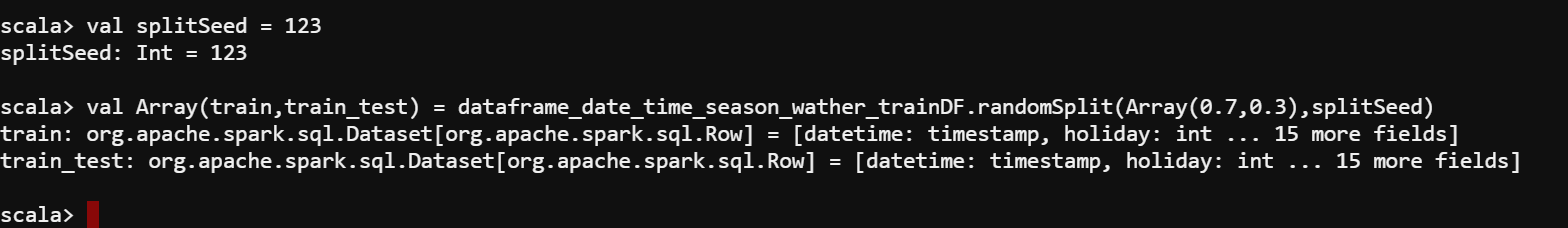
scala> val splitSeed = 123

splitSeed: Int = 123

scala> val Array(train,train\_test) = dataframe\_date\_time\_season\_wather\_trainDF.randomSplit(Array(0.7,0.3),splitSeed)

train: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [datetime: timestamp, holiday: int ... 15 more fields]

train\_test: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [datetime: timestamp, holiday: int ... 15 more fields]



2. Try different regression algorithms such as linear regression, random forest, etc. and note accuracy.

scala> val feature = Array(

| "holiday",

| "workingday",

| "temp",

| "atemp",

| "humidity",

| "windspeed",

| "season\_val",

| "weather\_val",

| "year",

| "month",

| "day",

| "hour",

| "minute"

| )

feature: Array[String] = Array(holiday, workingday, temp, atemp, humidity, windspeed, season\_val, weather\_val, year, month, day, hour, minute)

scala> val assembler = new VectorAssembler().setInputCols(feature).setOutputCol("features")

assembler: org.apache.spark.ml.feature.VectorAssembler = vecAssembler\_bf97e526af7a

=================================================================================

2.1Linear Regression Model.

//Model Building

val lr = new LinearRegression().setLabelCol("count").setFeaturesCol("features")

//Creating Pipeline

val pipeline = new Pipeline().setStages(Array(assembler,lr))

//Training Model

val lrModel = pipeline.fit(train)

val predictions = lrModel.transform(train\_test)

//Model Summary

val evaluator = new RegressionEvaluator().setLabelCol("count").setPredictionCol("prediction").setMetricName("rmse")

val rmse = evaluator.evaluate(predictions)

println("Linear Regression Root Mean Squared Error (RMSE) on train\_test data = " + rmse)

scala> println("Linear Regression Root Mean Squared Error (RMSE) on train\_test data = " + rmse)

Linear Regression Root Mean Squared Error (RMSE) on train\_test data = 143.53574705934028 lr: org.apache.spark.ml.regression.LinearRegression = linReg\_abae1ae92b8f pipeline: org.apache.spark.ml.Pipeline = pipeline\_07d792ea109d lrModel: org.apache.spark.ml.PipelineModel = pipeline\_07d792ea109d predictions: org.apache.spark.sql.DataFrame = [datetime: timestamp, holiday: int ... 17 more fields] evaluator: org.apache.spark.ml.evaluation.RegressionEvaluator = RegressionEvaluator: uid=regEval\_52841ecdbe07, metricName=rmse, throughOrigin=false rmse: Double = 143.53574705934028

2.2 GBT Regressor

val gbt = new GBTRegressor().setLabelCol("count").setFeaturesCol("features")

val pipeline = new Pipeline().setStages(Array(assembler,gbt))

val gbtModel = pipeline.fit(train)

val predictions = gbtModel.transform(train\_test)

val evaluator = new RegressionEvaluator().setLabelCol("count").setPredictionCol("prediction").setMetricName("rmse")

val rmse = evaluator.evaluate(predictions)

println("GBT Regressor Root Mean Squared Error (RMSE) on train\_test data = " + rmse)

GBT Regressor Root Mean Squared Error (RMSE) on train\_test data = 60.13502303606433 gbt: org.apache.spark.ml.regression.GBTRegressor = gbtr\_dcd84323fd9b pipeline: org.apache.spark.ml.Pipeline = pipeline\_0489f81b23a3 gbtModel: org.apache.spark.ml.PipelineModel = pipeline\_0489f81b23a3 predictions: org.apache.spark.sql.DataFrame = [datetime: timestamp, holiday: int ... 17 more fields] evaluator: org.apache.spark.ml.evaluation.RegressionEvaluator = RegressionEvaluator: uid=regEval\_0ba416fa9fcc, metricName=rmse, throughOrigin=false rmse: Double = 60.13502303606433

2.2 Decision Tree Regressor

//Model Building

val dt = new DecisionTreeRegressor().setLabelCol("count").setFeaturesCol("features")

//Creating Pipeline

val pipeline = new Pipeline().setStages(Array(assembler,dt))

//Training Model

val dtModel = pipeline.fit(train)

val predictions = dtModel.transform(train\_test)

//Model Summary

val evaluator = new RegressionEvaluator().setLabelCol("count").setPredictionCol("prediction").setMetricName("rmse")

val rmse = evaluator.evaluate(predictions)

println("Decision Tree Regressor Root Mean Squared Error (RMSE) on train\_test data = " + rmse)

Decision Tree Regressor Root Mean Squared Error (RMSE) on train\_test data = 108.42151766658162 dt: org.apache.spark.ml.regression.DecisionTreeRegressor = dtr\_ce718084f100 pipeline: org.apache.spark.ml.Pipeline = pipeline\_aa8f20abbd87 dtModel: org.apache.spark.ml.PipelineModel = pipeline\_aa8f20abbd87 predictions: org.apache.spark.sql.DataFrame = [datetime: timestamp, holiday: int ... 17 more fields] evaluator: org.apache.spark.ml.evaluation.RegressionEvaluator = RegressionEvaluator: uid=regEval\_2a4f3486b646, metricName=rmse, throughOrigin=false rmse: Double = 108.42151766658162

* 1. Random Forest Regressor

//Model Building

val rf = new RandomForestRegressor().setLabelCol("count").setFeaturesCol("features")

//Creating Pipeline

val pipeline = new Pipeline().setStages(Array(assembler,rf))

//Training Model

val rfModel = pipeline.fit(train)

val predictions = rfModel.transform(train\_test)

//Model Summary

val evaluator = new RegressionEvaluator().setLabelCol("count").setPredictionCol("prediction").setMetricName("rmse")

val rmse = evaluator.evaluate(predictions)

println("Random Forest Regressor Root Mean Squared Error (RMSE) on train\_test data = " + rmse)

Random Forest Regressor Root Mean Squared Error (RMSE) on train\_test data = 113.05487428850965 rf: org.apache.spark.ml.regression.RandomForestRegressor = rfr\_5823d31ef289 pipeline: org.apache.spark.ml.Pipeline = pipeline\_45ac2ab22c05 rfModel: org.apache.spark.ml.PipelineModel = pipeline\_45ac2ab22c05 predictions: org.apache.spark.sql.DataFrame = [datetime: timestamp, holiday: int ... 17 more fields] evaluator: org.apache.spark.ml.evaluation.RegressionEvaluator = RegressionEvaluator: uid=regEval\_95c4175ab92a, metricName=rmse, throughOrigin=false rmse: Double = 113.05487428850965

================================================================================

3.Select the best Model and Persist it

gbtModel.write.overwrite().save("/user/kaushikdey1984yahoo/modelSave/")

|  |  |
| --- | --- |
| Model Name | RMSE ( Root Mean Square) |
| Linear Regressor | 143.53574705934028 |
| GBT Regressor | 60.13502303606433 |
| Decision Tree Regressor | 108.42151766658162 |
| Random Forest Regressor | 113.05487428850965 |

So as we try diferent Regression Alorithms and found that "GBT Regressor Model" is giving better accuracy compare to other. And save it to the hdfs path : /user/kaushikdey1984yahoo/modelSave/

Graphical user interface, text, application, email

Description automatically generated

**Model Implementation & Prediction**

2.2 Application Development for Model Generation

For the above steps write an application to:

**1. Clean and Transform the data**

**2. Develop the model and persist it.**

**First we have to create BycyclePredict.scala file with logics and upload the FTP server and create the Jar Files from that. But we need to get following data.**

1. **Folder Structure where scala files will be present:**

[kaushikdey1984yahoo@ip-10-0-41-79 ByCycleModel]$ ls

[kaushikdey1984yahoo@ip-10-0-41-79 ByCycleModel]$ mkdir src

[kaushikdey1984yahoo@ip-10-0-41-79 ByCycleModel]$ cd src

[kaushikdey1984yahoo@ip-10-0-41-79 src]$ cd

[kaushikdey1984yahoo@ip-10-0-41-79 ~]$ cd ByCycleModel

[kaushikdey1984yahoo@ip-10-0-41-79 ByCycleModel]$ mkdir src/main

[kaushikdey1984yahoo@ip-10-0-41-79 ByCycleModel]$ ls

src

[kaushikdey1984yahoo@ip-10-0-41-79 ByCycleModel]$ cd src

[kaushikdey1984yahoo@ip-10-0-41-79 src]$ ls

main

[kaushikdey1984yahoo@ip-10-0-41-79 src]$ cd main

[kaushikdey1984yahoo@ip-10-0-41-79 main]$ mkdir scala

[kaushikdey1984yahoo@ip-10-0-41-79 main]$ ls

scala

[kaushikdey1984yahoo@ip-10-0-41-79 main]$ cd scala

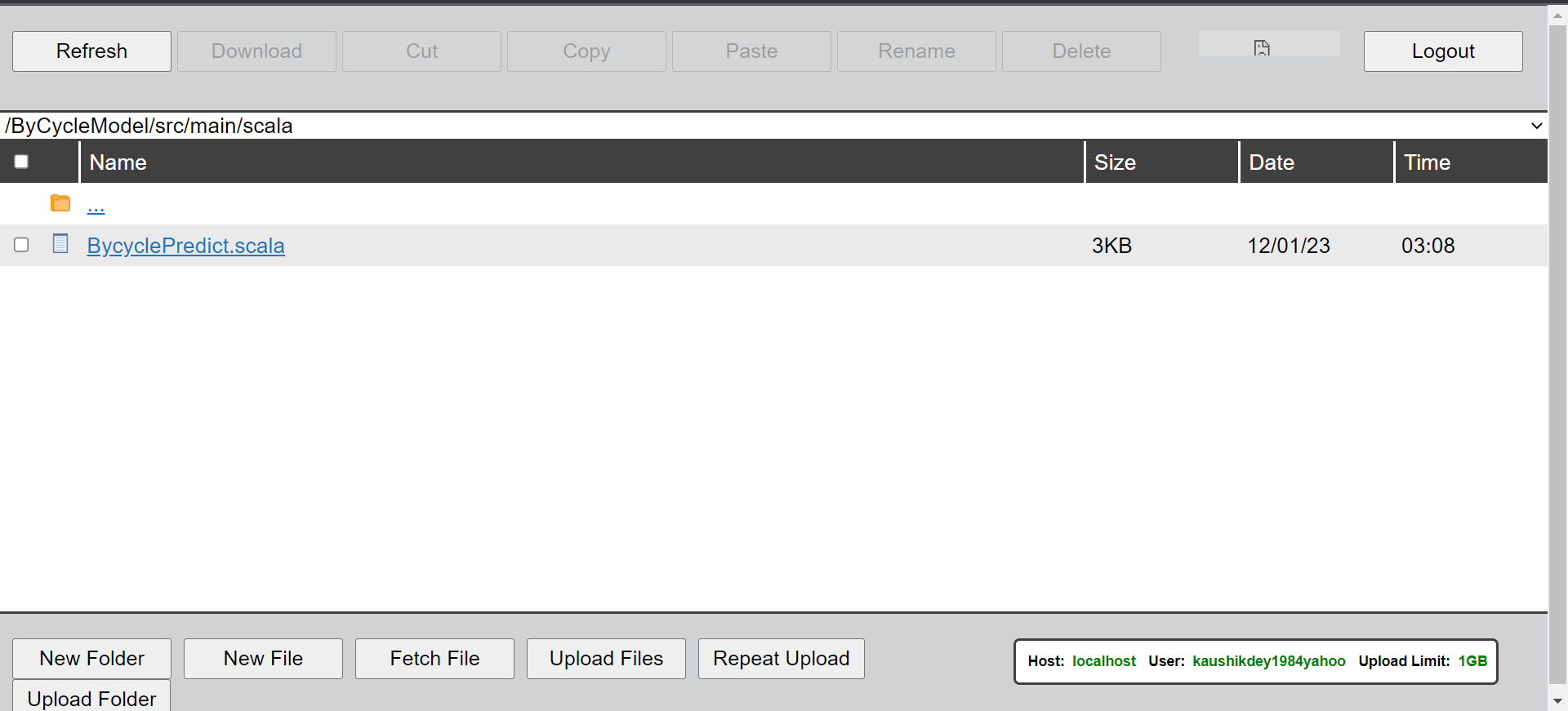
[kaushikdey1984yahoo@ip-10-0-41-79 scala]$ ls

[kaushikdey1984yahoo@ip-10-0-41-79 scala]$ ls

BycyclePredict.scala

[kaushikdey1984yahoo@ip-10-0-41-79 scala]$

1. **File Upload through FTP server.**



Now we must create build.sbt file for build the scala files and create the jar from it.

**Build.sbt :**

**name :="ByCycleModel"**

**version :="1.0"**

**scalaVersion :="2.11.8"**

**upload path:**

**/mnt/home/kaushikdey1984yahoo/ByCycleModel/build.sbt**

[kaushikdey1984yahoo@ip-10-0-41-79 ByCycleModel]$ ls

[kaushikdey1984yahoo@ip-10-0-41-79 ByCycleModel]$ mkdir src

[kaushikdey1984yahoo@ip-10-0-41-79 ByCycleModel]$ cd src

[kaushikdey1984yahoo@ip-10-0-41-79 src]$ cd

[kaushikdey1984yahoo@ip-10-0-41-79 ~]$ cd ByCycleModel

[kaushikdey1984yahoo@ip-10-0-41-79 ByCycleModel]$ mkdir src/main

[kaushikdey1984yahoo@ip-10-0-41-79 ByCycleModel]$ ls

src

[kaushikdey1984yahoo@ip-10-0-41-79 ByCycleModel]$ cd src

[kaushikdey1984yahoo@ip-10-0-41-79 src]$ ls

main

[kaushikdey1984yahoo@ip-10-0-41-79 src]$ cd main

[kaushikdey1984yahoo@ip-10-0-41-79 main]$ mkdir scala

[kaushikdey1984yahoo@ip-10-0-41-79 main]$ ls

scala

[kaushikdey1984yahoo@ip-10-0-41-79 main]$ cd scala

[kaushikdey1984yahoo@ip-10-0-41-79 scala]$ ls

[kaushikdey1984yahoo@ip-10-0-41-79 scala]$ ls

BycyclePredict.scala

[kaushikdey1984yahoo@ip-10-0-41-79 scala]$ cd ../../..

[kaushikdey1984yahoo@ip-10-0-41-79 ByCycleModel]$ ls

build.sbt src

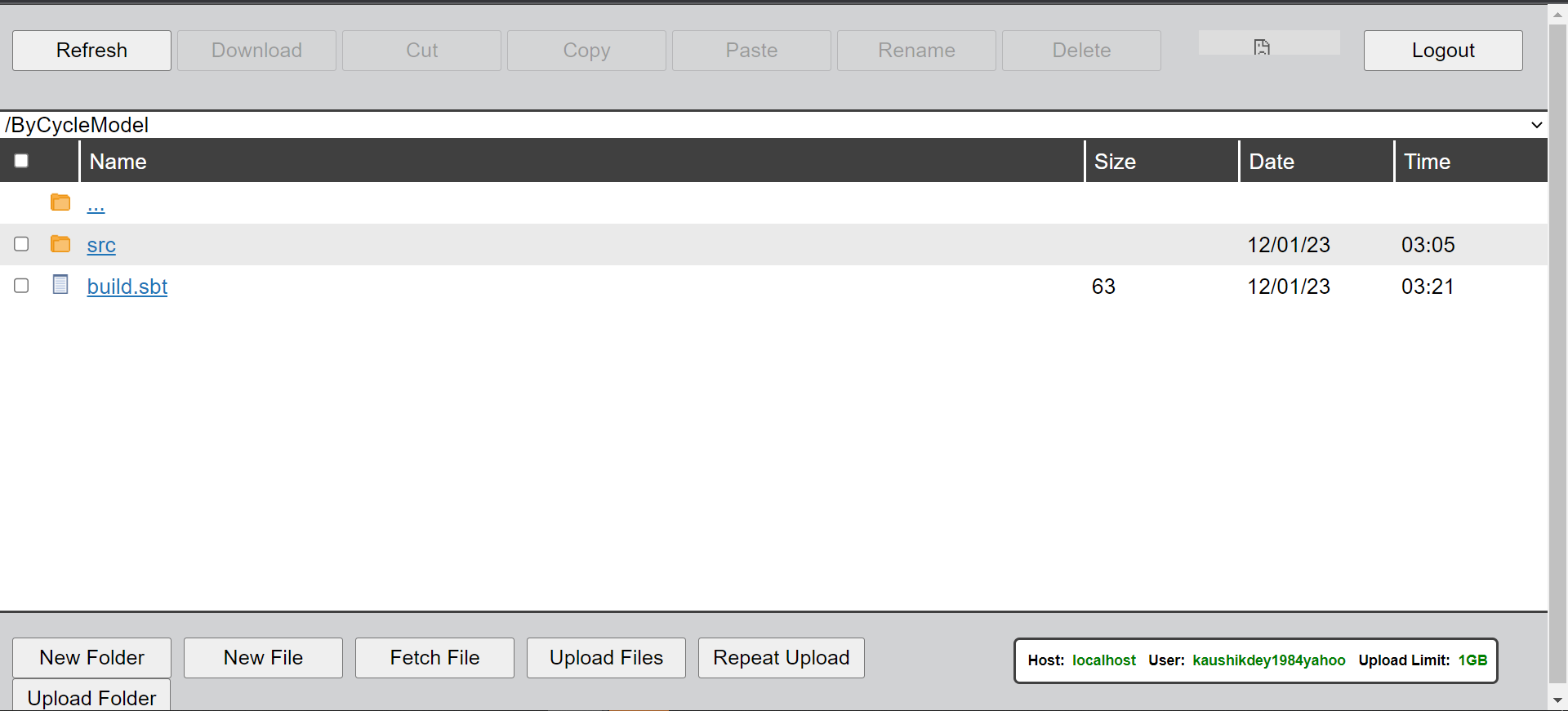
[kaushikdey1984yahoo@ip-10-0-41-79 ByCycleModel]$ ls -ltr

total 8

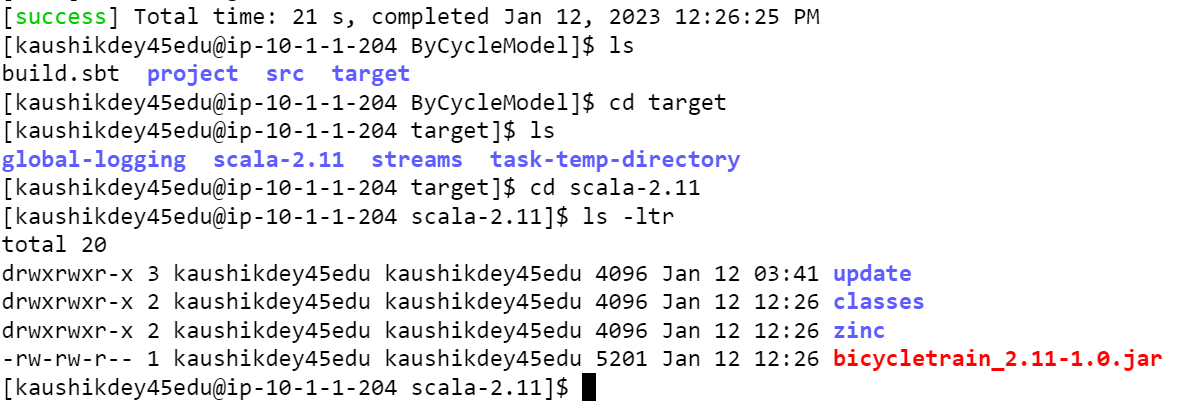
drwxrwxr-x 3 kaushikdey1984yahoo kaushikdey1984yahoo 6144 Jan 12 03:05 src

-rw-r--r-- 1 kaushikdey1984yahoo kaushikdey1984yahoo 63 Jan 12 03:21 build.sbt

[kaushikdey1984yahoo@ip-10-0-41-79 ByCycleModel]$



1. **Now Create the Jar Files.**



1. **Application Execution Mode.**

[kaushikdey45edu@ip-10-1-1-204 ~]$ spark-submit --class "BicyclePredictModel" --master yarn /mnt/home/kaushikdey45edu/ByCycleModel/target/scala-2.11/bicycl

etrain\_2.11-1.0.jar

Reading training data log

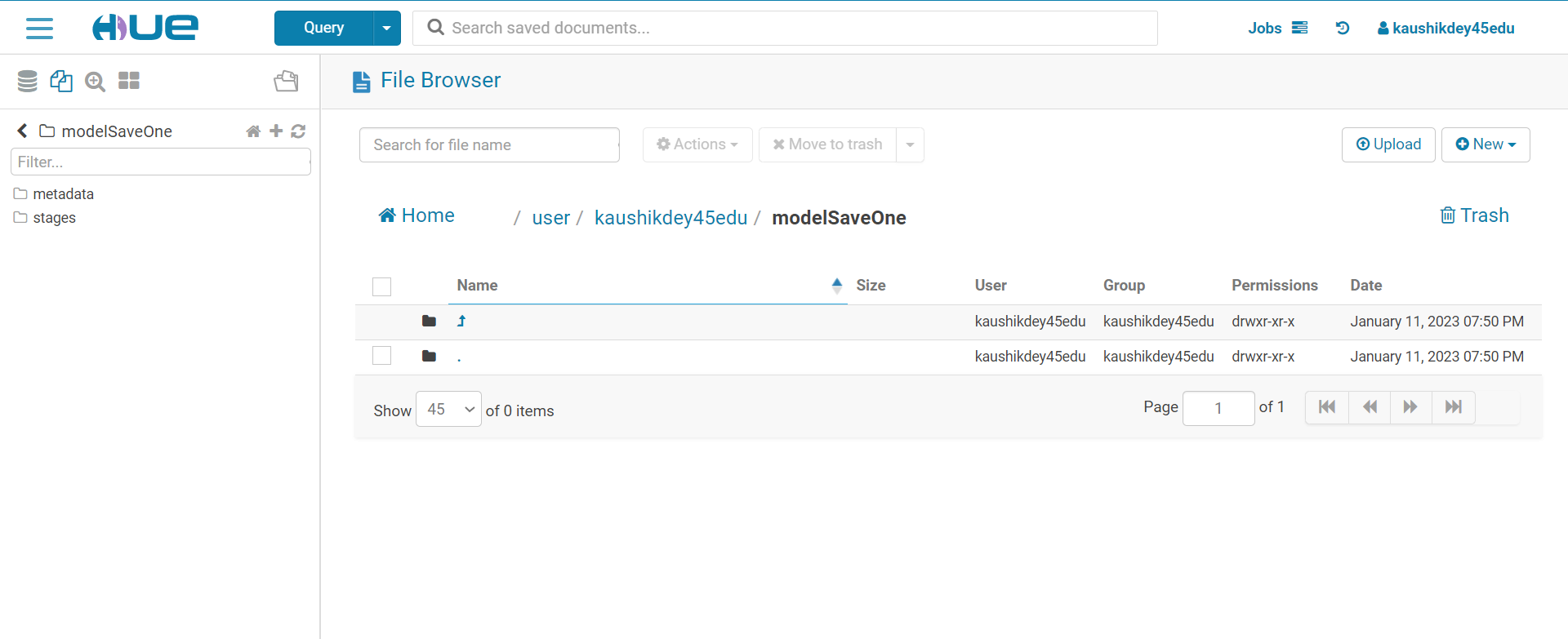
Cleaning data log

Training model.........................

**GBT Regressor Root Mean Squared Error (RMSE) on train\_test data = 59.39785790393261**

Persisting the model................

[kaushikdey45edu@ip-10-1-1-204 ~]$



**Application Development for Demand Prediction**

Model Prediction Application – Write an application to predict the bike demand based on the input dataset from HDFS:

**1. Load the persisted model.**

**2. Predict bike demand**

**3. Persist the result to RDBMS**

Entry Inside Mysql and create the table

**STEP 1**

[kaushikdey45edu@ip-10-1-1-204 ~]$ mysql -u kaushikdey45edu -p

Enter password:

Welcome to the MySQL monitor. Commands end with ; or \g.

Your MySQL connection id is 1679

Server version: 5.7.33 MySQL Community Server (GPL)

Copyright (c) 2000, 2021, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its

affiliates. Other names may be trademarks of their respective

owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>

**STEP 2**

mysql> CREATE TABLE predictionsModel (

->

-> datetime datetime,

->

-> count FLOAT

->

-> );

Query OK, 0 rows affected (0.01 sec)

mysql> select \* from predictionsModel;

Empty set (0.00 sec)

mysql>

**Prediction Application**

FileName:BycyclePredictTest.scala ( uploaded into the folder)

[success] Total time: 6 s, completed Jan 13, 2023 10:15:15 AM

[kaushikdey45edu@ip-10-1-1-204 ByCycleModel]$ sbt package

[info] welcome to sbt 1.5.5 (Oracle Corporation Java 1.8.0\_181)

[info] loading project definition from /home/kaushikdey45edu/ByCycleModel/project

[info] loading settings for project bycyclemodel from build.sbt ...

[info] set current project to bicycletrain (in build file:/home/kaushikdey45edu/ByCycleModel/)

[info] compiling 1 Scala source to /home/kaushikdey45edu/ByCycleModel/target/scala-2.11/classes ...

[warn] there was one deprecation warning; re-run with -deprecation for details

[warn] one warning found

[warn] multiple main classes detected: run 'show discoveredMainClasses' to see the list

[success] Total time: 6 s, completed Jan 13, 2023 10:22:47 AM

[kaushikdey45edu@ip-10-1-1-204 ByCycleModel]$

**Application Execution Mode**

spark-submit

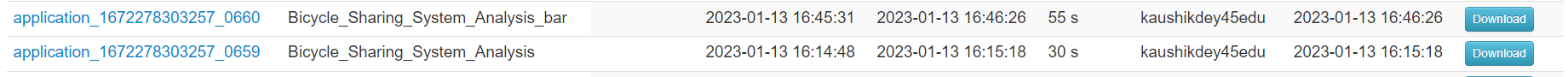
--packages mysql:mysql-connector-java:8.0.13

--class "BicyclePredictTest"

--master yarn /home/kaushikdey45edu/ByCycleModel/target/scala-2.11/bicycletest\_2.11-1.0.jar

**Spark History Server**

http://bigdata.cloudlabs.edureka.co:18088/



(eventLogs-application\_1672278303257\_0660\_bycyclesystemanalysisbar.zip as uploaded in project submission)

**Application For Streaming Data**

Write an application to predict demand on streaming data:

1. **Kafka Topic Creation . so first we have to delete the topic if it can be present in that code.**

**The command is that for**

kafka-topics

--zookeeper ip-10-1-2-175.ap-south-1.compute.internal:2181

--delete --topic kaushikdey45edu\_bicycle\_prediction

1. Now we have to create the topic for that

**Topic Creation**

kafka-topics

--create

--zookeeper ip-10-1-2-175.ap-south-1.compute.internal:2181

--replication-factor 1

--partitions 1

--topic kaushikdey45edu\_bicycle\_prediction

23/01/13 14:05:32 INFO zk.AdminZkClient: Topic creation Map(kaushikdey45edu\_bicycle\_prediction-0 -> ArrayBuffer(187))

Created topic "**kaushikdey45edu\_bicycle\_prediction**".

23/01/13 14:05:32 INFO zookeeper.ZooKeeperClient: [ZooKeeperClient] Closing.

23/01/13 14:05:32 INFO zookeeper.ZooKeeper: Session: 0xff855b8cd98c76db closed

23/01/13 14:05:32 INFO zookeeper.ClientCnxn: EventThread shut down

23/01/13 14:05:32 INFO zookeeper.ZooKeeperClient: [ZooKeeperClient] Closed.

ip-10-1-2-175.ap-south-1.compute.internal:2181 ( Server information got from Zookeeper Instances)

**kafka-producers**

kafka-console-producer

--broker-list ip-10-1-1-204.ap-south-1.compute.internal:9092

--topic kaushikdey45edu\_bicycle\_prediction

(ip-10-1-1-204.ap-south-1.compute.internal:9092) ( Kafka server Instances that I got from cloudera resource)

**kafka-consumers**

kafka-console-consumer

--bootstrap-server ip-10-1-1-204.ap-south-1.compute.internal:9092

--topic kaushikdey45edu\_bicycle\_prediction

--from-beginning

So, from console we can produce and read the data.

1. **Flume Configuration File**

ByCycleagent.sources = source1

ByCycleagent.channels = channel1

ByCycleagent.sinks = spark

ByCycleagent.sources.source1.type = org.apache.flume.source.kafka.KafkaSource

ByCycleagent.sources.source1.kafka.bootstrap.servers = ip-10-1-1-204.ap-south-1.compute.internal:9092

ByCycleagent.sources.source1.kafka.topics = kaushikdey45edu\_bicycle\_prediction

ByCycleagent.sources.source1.kafka.consumer.group.id = kaushikdey45edu\_bicycle\_prediction

ByCycleagent.sources.source1.channels = channel1

ByCycleagent.sources.source1.interceptors = i1

ByCycleagent.sources.source1.interceptors.i1.type = timestamp

ByCycleagent.sources.source1.kafka.consumer.timeout.ms = 100

ByCycleagent.channels.channel1.type = memory

ByCycleagent.channels.channel1.capacity = 10000

ByCycleagent.channels.channel1.transactionCapacity = 1000

ByCycleagent.sinks.spark.type = org.apache.spark.streaming.flume.sink.SparkSink

ByCycleagent.sinks.spark.hostname = ip-10-1-1-204.ap-south-1.compute.internal

ByCycleagent.sinks.spark.port = 4143

ByCycleagent.sinks.spark.channel = channel1

**Flume File location in edge-Node**

[kaushikdey45edu@ip-10-1-1-204 flumeAgent]$ pwd

/home/kaushikdey45edu/flumeAgent

[kaushikdey45edu@ip-10-1-1-204 flumeAgent]$ ls -ltr

total 4

-rw-rw-r-- 1 kaushikdey45edu kaushikdey45edu 1068 Jan 16 02:36 bycycle.conf

[kaushikdey45edu@ip-10-1-1-204 flumeAgent]$

**Run this Flume Agent**

flume-ng agent --conf conf --conf-file

/home/kaushikdey45edu/flumeAgent/bycycle.conf --name ByCycleagent

-Dflume.root.logger=DEBUG,console

**Configure spark streaming to pulldata from spark flume sink using receivers and predict the demand using model and persist the result to RDBMS.**

import org.apache.spark.{SparkConf, SparkContext}

import org.apache.spark.SparkContext.\_

import org.apache.spark.sql.\_

import org.apache.spark.sql.types.\_

import org.apache.spark.sql.functions.\_

import org.apache.spark.ml.regression.{GBTRegressionModel, GBTRegressor}

import org.apache.spark.ml.feature.{StringIndexer, VectorAssembler}

import org.apache.spark.ml.\_

import org.apache.spark.streaming.{Seconds, StreamingContext}

import org.apache.spark.streaming.flume.\_

import org.apache.spark.ml.Pipeline

import org.apache.spark.ml.feature.OneHotEncoder

object BicycleStreaming {

case class Bicycle(datetime: String, season: Int, holiday: Int, workingday: Int, weather: Int, temp: Double, atemp: Double, humidity: Int, windspeed: Double)

def main(args: Array[String]) {

val sparkConf = new SparkConf().setAppName("byCycleFlumeAgentConfig")

val sc = new SparkContext(sparkConf)

val ssc = new StreamingContext(sc, Seconds(2))

sc.setLogLevel("ERROR")

val spark = new org.apache.spark.sql.SQLContext(sc)

import spark.implicits.\_

val flumeStream = FlumeUtils.createPollingStream(ssc, " ip-10-1-2-175.ap-south-1.compute.internal", 4143)

println("Loading tained model.............")

val gbt\_model = PipelineModel.read.load("/user/kaushikdey45edu/modelSaveOne/")

val lines = flumeStream.map(event => new String(event.event.getBody().array(), "UTF-8"))

lines.foreachRDD { rdd =>

def row(line: List[String]): Bicycle = Bicycle(line(0), line(1).toInt, line(2).toInt,

line(3).toInt, line(4).toInt, line(5).toDouble, line(6).toDouble, line(7).toInt,

line(8).toDouble

)

val rows\_rdd = rdd.map(\_.split(",").to[List]).map(row)

val rows\_df = rows\_rdd.toDF

if(rows\_df.count > 0) {

val df\_time = rows\_df.withColumn("datetime",to\_timestamp(col("datetime"),"d-M-y H:m"))

val datetime\_testDF = df\_time.

withColumn("year", year(col("datetime"))).

withColumn("month", month(col("datetime"))).

withColumn("day", dayofmonth(col("datetime"))).

withColumn("hour", hour(col("datetime"))).

withColumn("minute",minute(col("datetime")))

//Onehot encoding on season nd weather column

val indexer = Array("season","weather").map(c => new OneHotEncoder().setInputCol(c).setOutputCol(c + "\_Vec"))

val pipeline = new Pipeline().setStages(indexer)

val df\_r = pipeline.fit(datetime\_testDF).transform(datetime\_testDF)

println("Making predictions...............")

val predictions = gbt\_model.transform(df\_r).select($"datetime",$"prediction".as("count"))

println("Persisting the result to RDBMS..................")

predictions.write.format("jdbc").

option("url", "jdbc:jdbc:mysql://ip-10-1-1-204.ap-south-1.compute.internal/kaushikdey45edu").

option("driver", "com.mysql.cj.jdbc.Driver").option("dbtable", "predictionsModelNew").

option("user", "kaushikdey45edu").

option("password", "PurpleCrow52@").

mode(SaveMode.Append).save

}

}

ssc.start()

ssc.awaitTermination()

}

}

Create the Table in MySQl , so we can sink the flume data in mysql

CREATE TABLE if not exists predictionsModelNew (

datetime datetime,

count FLOAT

)

Application Execution Mode

spark-submit

--packages mysql:mysql-connector-java:8.0.13

--class "BicycleStreaming"

--master yarn /home/ kaushikdey45edu/BicycleStreaming/target/scala-2.11/ bicyclestreaming\_2.11-1.0.jar

Output as CSV file Format

SELECT \* from predictionsModelNew

INTO OUTFILE '/user/kaushikdey45edu/flumeAgent/Submission.csv'

FIELDS ENCLOSED BY '"'

TERMINATED BY ';'

ESCAPED BY '"'

LINES TERMINATED BY '\r\n';

Push messages from flume to test the application. Here application should process and persist the result to RDBMS.

kafka-console-producer

--broker-list ip-10-1-1-204.ap-south-1.compute.internal:9092

--topic kaushikdey45edu\_bicycle\_prediction

(ip-10-1-1-204.ap-south-1.compute.internal:9092) ( Kafka server Instances that I got from cloudera resource)